Massachusetts Institute of Technology

Artificial Intelligence Laboratory

A.I. Memo 875

May 1986

Spatio-Temporal Reasoning

and

Linear Inequalities

Raúl E. Valdés-Pérez

Abstract

Time and space are sufficiently similar to warrant in certain cases a common representation in AI problem-solving systems. What is represented is often the constraints that hold between objects, and a concern is the overall consistency of a set of such constraints.

This paper scrutinizes two current approaches to spatio-temporal reasoning. The suitableness of Allen's temporal algebra for constraint networks is influenced directly by the mathematical properties of the algebra. These properties are extracted by a formulation as a network of set-theoretic relations, such that some previous theorems due to Montanari apply. Some new theorems concerning consistency of these temporal constraint networks are also presented.

It is argued that the linear programming approach to reasoning in time and space is needlessly complex, and is otherwise unsuitable as a submodule for a task that performs search.

In the spirit of the thematic tradeoff between expressivity and tractability, simple linear inequalities are adequately expressive and provide known algorithms which are amenable to a search regimen. Moreover, another known algorithm captures the real physical constraint of *disjointness*, and is therefore relevant to layout and planning tasks.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the research is provided partly by the Digital Equipment Corporation, and partly by the Office of Naval Research under contract number N00014-85-K-0124.

Table of Contents

List of Figures

1. Introduction

This memo examines the properties and purported advantages that some schemes, intended for spatio-temporal representation and inference, offer over simple linear inequalities and their algorithms. As such, this work belongs in the *implementation* slot of Marr's classification of AI research into the levels of computation, algorithm, and implementation [1]. Here, any discussion at the other levels in Marr's triad serves to point out that some current proposals at the implementational level were motivated by a particular task or computation to be achieved.

2. Time and Space in the Abstract

Many real-world applications of artificial intelligence require reasoning over time and space, but the *nature* of the reasoning leads to very different requirements. For example, a real skill such as that of a chess-player certainly demands an ability to handle spatial information, since a rook can capture its antagonist only if they both share one dimensional coordinate in the plane. A draftsman of electrical schematics must have a simple aesthetic appreciation of planar space in order to perform his task competently. Does the realization of both of these tasks by machine point toward a similar representation?

It so happens that although a human chess-player conceivably uses Euclidean distances on the planar chess board to evaluate features of a position, a mechanistic chess-player can eliminate the spatial property of the chessboard by making row and column indices into *nominal* (i.e. unordered) quantities, which are nevertheless still distinct. The fields of force of the chess pieces are *in theory* sufficient to discover the best move in a given position. It is unsurprising that the requirements that result from making space into a *nominal* quantity should differ from those requirements needed by our everyday notions of totally ordered spatio-temporal coordinates.

The postulate of special relativity offers a view of the world in which time and space are formally equivalent (if time is complex). However, the patrimonial *common sense* of most everyone reading this includes a world-view that treats time specially. We regard time as continually flowing in one direction at a constant rate, and furthermore distinguish that moment in time which is the *present*. This present moment halves all times into the past and the future. Any moment in this future is guaranteed to arrive, in the sense that the present will become that moment eventually and predictably.

Space, unlike time, is static according to our world-view. Neither do we discern a special location in this space that flows, bisecting space dynamically. These notions are familiar and commonsensical, and popularly regarded as the *natural* way to view these cosmic concepts. Could there be any other way that suitably explains everyday phenomena?

The linguist Benjamin Whorf concluded, by studying the Hopi Indian culture of Arizona, that their language had no notion whatsoever of time, either explicit or implicit, as a separate concept with the properties that we attribute to it [2]. However, as Whorf states,

> **... the Hopi language is capable of accounting for and describing correctly, in a pragmatic or operational sense, all observable phenomena of the universe.**

The Hopi metaphysics or world-view[1] imposes two forms that are comparable to our time and space: the manifested or objective, and the unmanifest or subjective. That which is *manifested* includes all that "is or

[1]Whorf uses the German term *Weltanschauung*, literally world-view.

has been accessible to the senses, the historical physical universe ... with no attempt to distinguish between present and past, but excluding everything that we call future." The *unmanifest* embraces our notion of future, together with the inobservable activity of the mind and of nature. The unmanifest is not advancing towards us as time is in our world-view, but is already present, although tending toward manifestation. Also included in the unmanifest is that realm of the manifested which is so distant (in our time and space) that it ceases to be knowable, and therefore re-enters the realm of the unmanifest or subjective. These two cosmic forms in the Hopi world-view pervade the grammar and verb forms of the language.

Although one can argue whether to lump together space and time, the discussion here will concern both to an extent for two reasons:

1. Some proposed representations claim to apply to either.
2. Space and time occasionally are represented uniformly. The fact that this uniformity is not universally desirable does not matter, because the previous example of chess and schematics shows that even for planar space quite distinct representations are in use.

The key properties that time/space share are order and a strictly increasing continuity (in the sense of the calculus). The equivalence, for our purposes, of space and time is due to the orthogonal Cartesian axes, which possess an order, and smoothly increase forever. A representation of space with spherical coordinates loses this continuity, and therefore the similarity with time.

3. Examples of Spatio-Temporal Assertions

In artificial intelligence, references to time/space arise from statements such as shown in the following list:

1. Ernest never wanders beyond where his beeper can reach him.
2. Man landed on the moon in 1969.
3. Her hangover lasted an entire day.
4. The hole in the wall is six inches above my desk.
5. Ted ran farther than Tom and Tod put together.

These statements can be intuitively translated into inequalities as follow:

1. $[\text{Ernest}_x - \text{beeper}_x]^2 + [\text{Ernest}_y - \text{beeper}_y]^2 \leq 10$ miles
2. $\underline{\text{Man landed on moon}} = 1969$ (years)
3. $\underline{\text{End her hangover}} - \underline{\text{Begin her hangover}} = 24$ (hours)
4. $\text{Hole}_z - \text{Desk top}_z = 6$ inches
5. $(\text{Ted}_{\text{finish}} - \text{Ted}_{\text{begin}}) > (\text{Tom}_{\text{finish}} + \text{Tod}_{\text{finish}}) - (\text{Tom}_{\text{begin}} + \text{Tod}_{\text{begin}})$

Equalities are expressible as a conjunction of two inequalities, so further discussion also treats the case of strict equalities also so here any discussion of the latter extends to the former.

Each of the translations except the first is a *linear inequality*. The first fails because it expresses a constraint which is not decomposable into independent constraints in the Cartesian plane. This non-linear inequality would be expressible as a linear inequality were our coordinate system polar. Each of the examples, except the first and last, is a *simple linear inequality*, which is of the form

$$q_i - q_j \geq a_{ij} \quad (1)$$

The last example is not a simple linear inequality because its equational form mentions more than two variables.

I therefore state the obvious fact that linear inequalities are incapable of capturing the full class of algebraic constraints, and that simple linear inequalities are less capable still.

Let us define a *program* as a set of algebraic constraints. Then another obvious but noteworthy fact is that the class of constraints expressible by a program of linear constraints (i.e. a *linear program*) subsumes or includes the class expressible by a program of simple linear constraints (a *simple linear program*).

4. Analyzing a Representation Scheme

Some useful questions to ask of a representation and inference scheme are:

Expressivity

- ***What facts are expressible in the scheme?*** The more the better, subject to a trade-off.
- ***What desirable facts are inexpressible?*** Disjunctive facts often are prohibited in part because of tractability, as in the terminological component of Krypton [3].
- ***Does the scheme cluster closely related objects?***

Deductive Power

- ***What "new" facts are deducible from the given facts?***
- ***What type of inconsistency is discovered?*** For example, a directed graph of arithmetic constraints is inconsistent whenever the sum of weights along the edges of a loop implies that a variable is greater than itself. Other task-dependent inconsistencies may be caught by the representation/inferential mechanism: no pair of variables may exist such that from each there is a zero-sum path to the other.
- ***Can the set of constraints be somehow inconsistent and escape detection?*** This often reflects a conscious trade-off on the part of the designer, whenever consistency is achievable but at high cost.
- ***What information about inconsistencies is reported?*** Are nogoods reported, and are they minimal or nearly-minimal sets?
- ***What is the computational cost of reporting inconsistencies?*** Occasionally, nogood sets are a by-product of the detection of the contradiction, so the incremental cost is nil.
- ***What is the computational cost of the deduction of "new" facts?***
- ***Does the inferential mechanism deduce many useless facts?*** One of the purposes of clustering is to avoid this very problem.

Implementational Congeniality

- ***Is there more than one representation for similar facts, or is the representation uniform?*** Uniformity is a virtue, but some tasks are served better by different representations.
- ***Is the scheme incremental (does it promote search), in the sense that it is cheap to add and delete constraints?*** A program module promotes search if the addition of a few constraints to a "solved" system, and then solving, is cheaper than adding those constraints

to an unsolved system and solving everything together.

- ***If the scheme is intended as a module in a larger problem-solver, is the representation in harmony (with regard to data structures) with other representations within the problem-solver?*** Lack of harmony means more clutter for the programmer to comprehend.
- ***What is the programming complexity of the algorithms which carry out the functions of the scheme?*** A good example of a complex algorithm is Simplex (discussed further below).

In order for a scheme to solicit acceptance, it should offer some newly advantageous answers to some of the questions above. In what follows, I shall treat some current approaches by examining these questions.

5. Allen's Interval-Based Representation

5.1. Historical Origin

Allen over the past few years has proposed a scheme [4, 5] for representing temporal assertions. This scheme seems historically motivated by the reasoning tasks typical of the understanding of narratives, which is the domain of his doctoral thesis. His representation has further evolved as part of a larger effort to design a logic that captures propositions, events, actions, and plans that are qualified temporally.

To the extent that this algebra of temporal relations is advocated for multiple tasks of reasoning and problem-solving [6], we are justified in examining closely its properties.

5.2. What Problem Does This Scheme Solve?

I view Allen's scheme as essentially answering the following question:

> **Problem 1:** How should one design a mechanism to infer new facts about the relationships between pairs of events, given ambiguous knowledge that holds between some of the pairs?

The key aspects of this formulation are:

- An emphasis on the capacity to infer new facts, and therefore on the soundness of the inference.
- The use of disjunctive facts which are "local" in the sense that they make a statement about the relationship between *two* events.

A temporal-reasoning task that lacks these aspects but emphasizes others requires a different approach. Other plausible aspects are:

- The mechanism is consistent; it discovers tractably the existence of contradictory sets of relations.
- Disjunctive facts are unnecessary; instead a set of assertions is strictly conjunctive.[2]

In what follows, I shall present in more detail Allen's approach, and show how it adequately solves Problem 1. Moreover, this paper examines the suitableness of the approach as a solution to the following problem:

> **Problem 2:** What representation, inferential mechanism, and guarantor of consistency are appropriate for a task which repeatedly *proposes* relationships between variables (e.g. events) in the search for a satisfactory arrangement?

Some other approaches will be examined in light of this question.

---

[2]Of course a task may involve proposing different conjunctions, as during search. Such behavior is analogous to searching a disjunction. In that case, the conjunctiveness refers to some subordinate procedure, which is a "snapshot" of the global search task.

5.3. Review of the Scheme

Allen introduces the notion of an *interval*, which is not crisply defined, but is vaguely a one-dimensional thing which has a nonzero length. These intervals are meant to correspond to events, so one extremum of the interval is "greater than" the other, in the sense that there is exactly one correct orientation of this interval along the time dimension. Hereafter I shall talk about "endpoints" of intervals, ignoring the philosophical and logically rigorous issues involved, which are not relevant to the present purpose.

Representable facts about intervals are the relative position of *two* intervals with respect to the orientation of time. Furthermore, any uncertainty in their relative position is captured by a *set* of relative positions, which implies a disjunction. So for uniformity, a relationship between intervals is always a set, possibly of a single element.

The vocabulary of possible relationships follows:

before (<) ↔ after (>)
=
overlaps (o) ↔ overlapped-by (oi)
starts (s) ↔ started-by (si)
finishes (f) ↔ finished-by (fi)
during (d) ↔ contains (di)
meets (m) ↔ met-by (mi)

Two relations on a same line above are inverses, e.g. if interval A is *during* interval B, then B *contains* A. Between intervals there are thirteen possible relative positions, all of which are mutually exclusive. These thirteen are easily generated by distinguishing the endpoints of two example intervals and listing the possible relative positions among their four endpoints.

Given a representation of events as intervals, and relationships between intervals as sets of relative positions, one still needs a *mechanism* to make explicit those relationships which are initially only implicit. The transitive operation is that deductive mechanism, and to that end Allen straightforwardly defines a transitivity table, which is reproduced in Appendix I. This table defines the result of the composition of two relative positions, for example:

{<} ° {<} = {<}
{<} ° {d} = {< o m d s}
{m} ° {<} = {<}
{m} ° {d} = {o d s}

Transitive inference is accomplished by taking the Cartesian product of two relationships (i.e. sets), and composing each of the resulting pairs according to this table. The union of the sets obtained by composing the pairs is the new inferred relationship. Continuing the example above,

if A *{< m}* B **and** B *{< d}* C **then** A *{< o m d s}* C.

The mutual exclusivity of the relative positions is what allows performing simply the union to obtain the new relationship. I reiterate the interpretation of these sets as a disjunctive assertion regarding the relative position between two intervals.

Finally, the notion of intervals and their relationships are embedded in a directed graph (i.e. a network). In conformity with the suggestion contained in Patrick Hayes' papers, the following account makes explicit the semantics of an interval network:

A particular instance of a network asserts the conjunction of the relationships implied by the edges between the vertices (i.e. the intervals). Any two vertices between which no edge exists is

considered to have the universal relation (i.e. the set of all relative positions).

The network asserts a conjunction of many disjunctions. However, this conjunction is not minimal because there are implicit relationships therein whose determination would reduce the size of embedded disjunctions. This determination of implicit relationships is exactly what the *transitive closure* algorithms intend; more on that below.

In my view, the key idea of this scheme is that an interval is not described as a pair of endpoints, but as an indivisible entity. In this way, disjunctive uncertainty is included in the relationship between interval pairs, as befits the constraint propagation paradigm. Incorporating similar disjunctions to a point-based representation is clumsy, and is incompatible with the idea of local constraint propagation.

5.4. Constraint Networks and Their Properties

This section reviews a theory of networks of constraints, after which we can analyse Allen's scheme by casting it such that the theory applies. The formulation and theorems are due to Montanari [7].

Allen and Hayes have recently axiomatized the notion of intervals and their relative positions [8]. Intervals are not defined, rather they are characterized by their properties. Relative positions are defined by existential quantification over intervals and the primitive relative position *meets*. Their axiomatization allows a discrete time model, and it is this model to which the analyses here relate. No theorematic statement about a continuous time model shall be made.

In Allen's scheme, the relationship between intervals is a set of relative positions, up to thirteen in cardinality. Assuming a discrete time model, the set of all possible intervals constitutes an enumerable and totally ordered set. For example, if the number of time instants is a finite N, then the number of possible intervals is N(N-1)/2, and the intervals are orderable according first to the lesser point, and then to the greater point.

The use of the term *interval* misleads somewhat, because it suggests a known quantity in time. If these quantities are known, then there is no ambiguity concerning their relationship, whereas here we treat clearly uncertain relationships between intervals. Not wishing to introduce more terminology, instead I stress that an interval is an unknown quantity in time, in other words, a *variable*.

A relationship between two intervals, therefore, is viewable as a mathematical *relation*: it is a *subset* of the Cartesian product of the domain sets of the intervals. Those pairs that are in the subset are allowed by the constraint, otherwise they are precluded. Then a *network* is defined here, borrowing from Montanari, as:

***A network R of binary relations* is defined as a set of sets $X = \{X_1,...,X_n\}$ plus a relation R_{ij} from every set X_i to every set X_j (i,j=1,...,n).**

It is this view of interval networks which eases the proof of new theorems regarding them.

There are two concepts needed to understand the theorems of Montanari:

- A closed network.
- A minimal network.

Since Montanari's paper contains precise formalizations of these concepts, I shall here give only an intuitive interpretation. A **closure**, or a **closed network**, is a network in which all implicit (i.e. along a path) relations between intervals have been made *explicit* by including their effect in the direct arc-constraint between the intervals.

A **minimal network** is the *smallest* of the set of closed networks which are equivalent to the original, possibly unclosed, network. *Smallest* refers to a partial order on networks of equal number of vertices. The interesting property of a minimal network is the following paraphrase of Montanari's theorems:

> **Theorem 1:** (Montanari) A network is minimal if and only if for any vertices i and j, if any pair of values for i,j satisfies the direct relation between them, then that pair of values appears in some solution (i.e. n-tuple) that satisfies the overall network.

It is crucial to grasp the difference between a closed network and a minimal network.

A simple example which highlights the difference is that of a network that is internally path-consistent, which nevertheless possesses no assignment to the vertices that fails to contradict some relation. One can not know whether a network has a consistent assignment to its vertices by solely examining its connectivity and the relationships along the edges; the domains of the vertices are also needed.[3]

The graph-coloring example on page 10 illustrates this idea. The network shown is already closed, i.e. there is no path-wise constraint which is not already explicit in the direct binary relations. By varying the domain of the vertices, as in cases A) and B), we see that the case B) *does* indeed possess a solution in which each vertex receives a different color. Because there is no solution to case A), its minimal network would be that network with the null relation on every arc.

The central problem, according to Montanari, is computing tractably the minimal network equivalent to a given network. It would be nice if the closure of a given network were provably minimal. Montanari proves that the closure is minimal for two classes of networks, both of which are sparsely connected:

1. Symmetric tree networks.
2. Symmetric series-parallel networks.

An example of a symmetric tree network is a tree for which the relation along each branch is the "not equal" relation. Such a network is symmetric because if a pair of values {a,b} for two connected vertices is prohibited, then the pair {b,a} is also prohibited.

Unfortunately, such networks are too limited to serve usefully in the domain under discussion. Another way to guarantee that the closure be minimal is to restrict the type of relation that holds between vertices. Again, however, the restrictions necessary are too stringent to be useful. For example, the relation "equals" between two variables does *not* meet the restrictions.[4]

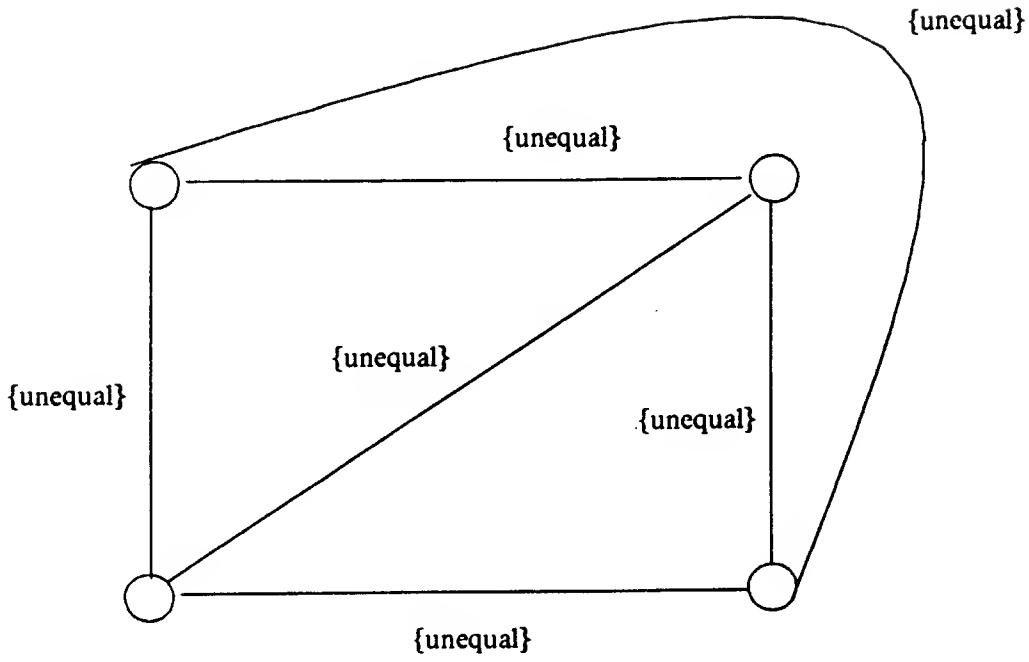
5.5. Consequences for Allen's Scheme

The lesson from the previous mathematical observations is that neither the closure nor the similar constraint propagation algorithm, given by Allen [4] for the addition of a new constraint to a network, is *consistent*. The closure cannot reveal whether a network has an assignment to its vertices which fulfill the binary relations. Moreover, there is scant hope that such a scheme can be made consistent, in view

---

[3] If, instead of a *relationship* "≠", we had the corresponding *relation* which consists of all pairs from each variable that satisfy "≠", then it is possible to determine whether there is a consistent assignment.

[4] The reason is that the relations must be, to use Montanari's term, monotone. So if an element χ from the domain of X is included in a relation with element ψ from the domain of Ψ, then all domain elements less/greater than χ, according to the partial order defined over the domain elements, must also be included in a relation with ψ.



A) With a domain of {red white blue}, there is no consistent coloring of the graph.

B) With a domain of {red white blue pink}, there IS a consistent coloring.

Figure 5-1: The Graph-Coloring Example

of the above. Given a closed network, and assuming that the number of possible intervals in the domain is large (or expandable) so as to avoid the difficulty shown in the graph-coloring example, how can one otherwise determine the consistency of the network?

The common recourse is to search through the space of possibilities. One space is the domain of the possible intervals: repeatedly assign a chosen interval to the network vertices until reaching a configuration which satisfies the disjunctions on the edges. If the domains are large, a better approach is to search the space of singleton subnetworks, by choosing a single disjunct (relative position) on each arc, and then testing its consistency. For example, the network on page 12 gives rise to six singleton subnetworks, which are also shown.

The key question then becomes:

Can a constraint-propagation operation using Allen's transitivity table guarantee consistency in singleton networks, or must one escape to another representation in order to confirm the existence of a solution?

It is easy to see that the closure of a singleton network is either the same network or the null network, because the closure computes a (not necessarily strict) subnetwork of the original network. However, we have already shown that the disjunctive scheme and transitivity table, together with local constraint propagation, does not guarantee consistency. Montanari's theorems furnish ready cases for which the closure is minimal, but the preconditions don't apply here either.

Fortunately, since singleton networks are *not* disjunctive, it is possible to prove that their closure is *minimal*. For that we yet need a theorem regarding closed paths in interval networks, which is useful in its own right and will be derived shortly.

In the remainder of this section, I shall prove a theorem concerning Allen networks, and show how the theorem can heuristically prune the search for a consistent subnetwork of an original closed network. Before proceeding, I shall list the interesting observations from this entire section:

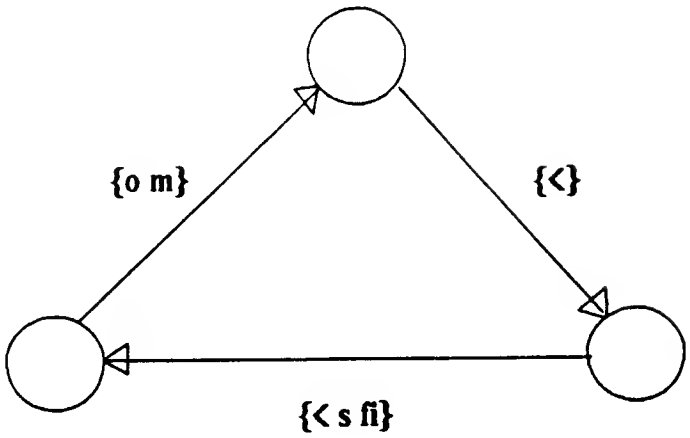
- Allen's deductive mechanism is sound, but not consistent. Neither the closure nor his constraint propagation algorithm can guarantee the existence of a solution satisfying the edge constraints.
- To determine whether a solution exists, one can perform an exponential search through the disjunctions, i.e. testing all singleton subnetworks of the closure of the given network.
- The closure of a singleton network will be null if the network is unsatisfiable.
- In a closed non-null Allen network, "=" is an element of the composition of relations along any closed path.
- As a heuristic[5] during the search for a consistent singleton, one can prune a proposed assignment to an edge if it would result in a loop (say, of length three) which does not include the "=" element.

In what follows I prove these last three observations.

I have already shown how Allen's temporal networks are formulable as networks of set-theoretic *relations* over the domains of the vertices (intervals). Such pair-wise relations are a subset of the Cartesian product of the domains; alternately they are matrices whose ij element is unity only if the pair ij is allowed by the binary relation. The composition of relations is then equivalent to matrix multiplication (with Boolean arithmetic), which is certainly associative. So the first result is:

---

[5]With the meaning of "more informed than blind search."



Singleton Subnetworks

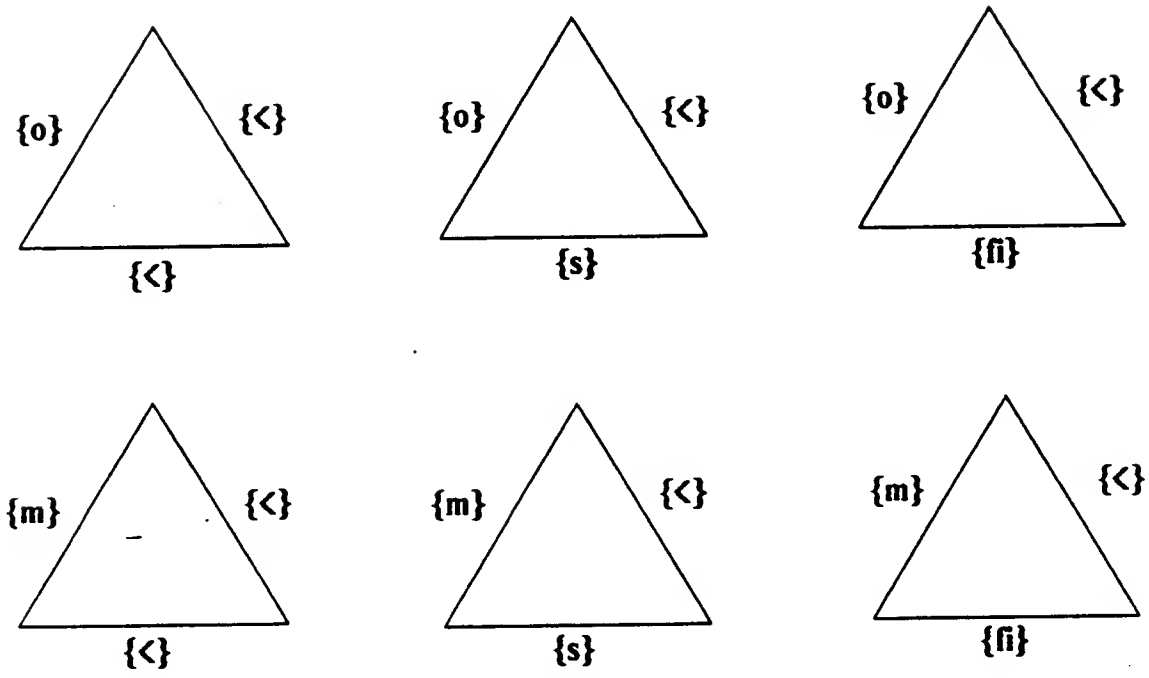


Figure 5-2: Singleton Networks

Lemma 2: Allen's algebra of intervals is associative.

Now let ***inv*** be a function which takes the inverse of the input relative position, for example ***Inv***(<) = >. Let the capitalized ***INV*** do the same for all the elements of a set (e.g. ***INV***({> o fi}) = {< oi f}. One can easily prove a result that is suspiciously similar to the theorem from matrix algebra:

$$[M_1 M_2]^{-1} = M_2^{-1} M_1^{-1}$$

To prove the analogue, let *table*(r_1,r_2) denote the function corresponding to Allen's transitivity table, which outputs a set. Then by exhaustive enumeration (13^2 cases) one obtains that:

Lemma 3: $\boldsymbol{INV}\langle table(r_1, r_2)\rangle = table(\boldsymbol{inv}(r_2), \boldsymbol{Inv}(r_1))$

which permits the following result:

$$R_1 \circ R_2 \equiv \cup_{j,k} table(r_{1j}, r_{2k})$$

$$\boldsymbol{INV}\langle R_1 \circ R_2\rangle = \boldsymbol{INV}\langle \cup_{j,k} table(r_{1j}, r_{2k})\rangle$$

$$= \cup_{j,k} \boldsymbol{INV}\langle table(r_{1j}, r_{2k})\rangle$$

$$= \cup_{j,k} table(\boldsymbol{Inv}(r_{2k}), \boldsymbol{Inv}(r_{1j})) \quad \text{;lemma 3}$$

$$= \boldsymbol{INV}\langle R_2\rangle \circ \boldsymbol{INV}\langle R_1\rangle$$

and the theorem

Theorem 4: Using Allen's transitivity table, where R denotes a binary relation,

$$\forall j,k \ \boldsymbol{INV}\langle R_j \circ R_k\rangle = \boldsymbol{INV}\langle R_k\rangle \circ \boldsymbol{INV}\langle R_j\rangle$$

Before proving the main theorem of this section, another result is needed:

Lemma 5: If an Allen network is closed and non-null, then "=" is an element of the composition of the relations along every closed path of length three.

Proof. Assume that the network is closed and non-null. By definition of a closed network, we know that $\forall i,j,k \ R_{ij} \subseteq R_{ik} \circ R_{kj}$. Since the network is non-null, there must exist a relative position which is an element of both R_{ij} and $R_{ik} \circ R_{kj}$. Therefore, the inverse of that element is included in R_{ji}. If ***Inv***(r) = s, then by inspection of the transitivity table *table*(r,s) includes "=", which is therefore included in the closed path $R_{ik} \circ R_{kj} \circ R_{ji}$.

Q.E.D.

We now have sufficient tools with which to obtain a main theorem:

Theorem 6: If an Allen network is closed and non-null, then "=" is an element of the composition of the relations along every closed path.

Proof. We consider an arbitrary loop of length *m*, and construct a corresponding polygon having *m* sides. Distinguish a vertex of the polygon as that vertex which starts and ends the closed path. Now connect the distinguished vertex to all other vertices of the polygon. This creates *m*-2 inscribed triangles in the polygon. Selecting a smallest triangle, we know from the previous lemma that the composed relations along it include "=." Now successively replace a relation on one of its faces by the similar path on an

adjacent triangle, until one sweeps the entire polygon, obtaining finally the closed path that touches each vertex once, and the end vertex twice. At teach step the larger path also includes "=", because the network is assumed closed so that each two-step path that replaces a single relation is a superset of this relation.

Q.E.D.

As promised, with Theorem (6) we may prove the following:[6]

Theorem 7: These statements are true and equivalent:

1. The closure of a singleton Allen network is minimal.
2. A closed Allen network with a single non-null relative position on every edge has a solution.
3. If all triangular subnetworks of a singleton non-null Allen network are consistent, then the network has a solution.

Proof by contradiction. Assume a closed singleton non-null Allen network that *does not* possess a solution.

Imagine a new network that splits intervals into their two endpoints. This network translates the interval relations by connecting two nodes in the endpoint network by a unidirectional red edge to denote the predicate "precedes," and by a bidirectional white edge to denote the predicate "equals." The *meaning* of these two networks is identical.

Now this split network is inconsistent only if there is a closed path of edges that includes at least one red edge, because otherwise the network is simply a partial order which of course is consistent. Distinguish, without loss of generality, a vertex A_{low} on the closed path that corresponds to the lower endpoint of *interval* A. We know from the existence of the loop that $x_{low}>y_{low}$, where in fact both x and y are A.

From this closed path in the split network, we can obtain the corresponding closed path in the Allen network that mentions the same intervals. In particular, the composition of relations along this path, starting and ending at interval A, must admit $x_{low}>y_{low}$, where x=y=A. The only relative positions between x and y that admit this constraint are {< di o m fi}. Therefore the composition of relations along that closed path beginning and ending at A in the Allen network is a subset of, or equal to, {< di o m fi}. Because this set does not include the element "=", we note, on account of Theorem (6), a contradiction.

The three equivalent statements of the theorem above are obtained simply by replacing terms by their definitions or trivial consequences.

Q.E.D.

Another consequence of Theorem (6) is that, when doing a simple depth-first search for a consistent singleton network from a larger disjunctive network, a next assigned relation to an edge can be retracted if any loop of length three would be formed whose composed relations exclude the "=." An upper bound on the order of the total work done by applying such a heuristic during a serendipitously backtrack-free path is:

$$C(2,2) + C(3,2) + C(4,2) + \ldots + C(V\text{-}1,2) = C(V\text{-}1,3) \Rightarrow O(V^3)$$

---

[6]The interval *variables* are assumed to take on values from a domain extending indefinitely, or at least large enough so that a situation analogous to that in the figure on page 9 does not arise.

where C denotes combinations and V is the number of vertices.[7] The equality is from a known combinatorial identity. Potentially at each step an exponential amount of work can be saved, although the exponent decreases as more edges are labelled with a relation.

5.6. Practical Aspects of the Scheme

This section treats informal aspects of Allen's scheme which influence its fitness for general problem-solving. These aspects involve the expressivity and the handling of contradictions.

It is easy to see that any single relative position between intervals is a conjunction of two algebraic equations of the particularly simple form:

$i < j$ ***or*** $i > j$ ***or*** $i = j$

The interval-based scheme as proposed, however, does not permit information about *distances* between intervals. One can incorporate distances to the point-based scheme in (2) by adding a constant to the (in-)equalities. This degrades only slightly the ease of determining the consistency of such a set of (in-)equalities, from a topological sort to a shortest-path algorithm (more on that later). Therefore the interval-based representation is less expressive than a point-based setup, in the sense of disallowing distances. However, Allen networks naturally capture disjunctive facts, whereas a point-based network does not. In my view, this key difference makes the two schemes not directly comparable.

Another vital matter concerns how to exploit the occurrence of an inconsistency by reporting and storing *nogood* sets, i.e. a preferably minimal set of constraints which by themselves account for a contradiction. There are two cases to consider, because there are two ways that inconsistencies arise. First, a contradiction is uncovered when, during constraint propagation, an edge remains with a null relation. Conceivably, some mechanism for truth maintenance could handle expediently the building of nogood sets, and in fact this author has been informed of a current project at BBN with this goal [9]. I conjecture that any nogood set obtainable in this case corresponds exactly to a closed path whose composed relations do not include "=."

Recall that some closed non-null Allen networks are not satisfiable, and that the recourse then is to exponentially search for a consistent singleton subnetwork. If no such singleton exists, then evidently the original network is inconsistent, and we have the second type of contradiction hinted at above. The resulting nogood set is the entire original network, which is not very useful.

6. Full Linear Programming

Malik and Binford describe [10] a scheme which consists of representing constraints as linear inequalities in any number of variables. It then uses the Simplex algorithm to make "deductions" on these inequalities. I shall analyze this scheme by answering the questions previously listed regarding expressivity, deductive power, and implementational congeniality.

With regard to **expressivity**, their scheme handles spatial constraints by formulating relationships between points in space as linear inequalities. Time is similarly represented, by considering events to be intervals characterized by a beginning point and an ending point on a time line. Since their scheme is full

---

[7]The author has derived, as a combinatorial exercise, the exact expression $\Sigma\, i(i-1)/2$ for $i=2,V-1$.

linear programming, the problematic previous statement

Ted ran farther than Tom and Tod put together.

$$(Ted_{finish} - Ted_{begin}) > (Tom_{finish} + Tod_{finish}) - (Tom_{begin} + Tod_{begin})$$

is expressible, although the other problematic statement

Ernest never wanders beyond where his beeper can reach him.

$$[Ernest_x - beeper_x]^2 + [Ernest_y - beeper_y]^2 \leq 10 \text{ miles}$$

is not expressible, due to the quadratic terms. The ability to cluster semantically related information is claimed, while relating these clusters by a "reference frame transformation" which apparently amounts to stating, for example, that *cluster-1* is ten meters after *cluster-2*.

I now examine the **deductive power** of the Malik/Binford scheme. The basic deductive step is the transitivity operation, which depends on the transitivity of the "less-than" relation. As the authors mention, an interesting relationship between two quantities, call them *A* and *B*, is whether *A* is necessarily less than *B*. Assuming that an explicit relationship between them is absent, then simply executing the Simplex algorithm and obtaining numbers for *A* and *B* is insufficient, because one quantity may be assigned a larger number *casually*, not causally. Since their scheme allows this query, some further computation beyond the Simplex step is needed, since Simplex doesn't reveal information such as "*A*<*B* in any solution of the constraints."

The possibility that a set of linear inequalities is inconsistent can be detected by a properly implemented Simplex program (there are other complications due to the Simplex algorithm itself, but those are surmountable). The authors state that the "system refuses to accept a constraint that is inconsistent with the previous set", and that the system accomplishes this detection without running the full Simplex algorithm.

No mention is made of what, if any, information is reported when a contradiction arises. The given framework does not seem to support the reporting of nogood sets. It is not apparent from the paper what the computational cost of "deducing" new facts is, because, as said, the Simplex algorithm doesn't really account for this ability. It is also unclear how well this scheme *promotes search*, by the aforementioned definition.

The most vexing aspect of Simplex is that its complexity is exponential. Determining solely the consistency of a set of inequalities involves, in the version of Simplex seen by this author [11], the execution of the entire algorithm just to find an initial basic feasible solution.[8] The failure to find this initial basic feasible solution signals the inconsistency of the inequalities. Since search is characteristically driven by failure, this observation speaks poorly for using linear programming in a problem-solver.

From a programmer's view, another drawback of the linear programming approach is the lack of **implementational congeniality**. Although the representation is uniform, it involves ugly data structures which do not easily integrate with the rest of a problem-solver. Integration is not necessary, but if much information can be "hung" on the same data structures, then the conceptual complexity of the program is less.

Finally, the cognitive complexity of the Simplex algorithm is quite distressing. When one deals only

---

[8]This procedure does not recurse infinitely because, apparently, finding an initial basic feasible solution for this subordinate linear programming problem is automatic.

with linear inequalities in two variables, and is not interested in maximizing some *objective*, then the use of Simplex amounts to overkill, since simpler and more intuitive algorithms are known for this case. The overriding purpose of Simplex is to maximize an *objective* function (some linear algebraic expression in the variables) subject to some constraints, so that its use for constraint satisficing seems superfluous. For this same reason, one would not use MACSYMA[9] solely to integrate polynomials.

7. Simple Linear Programming

This author claims that an appealing approach to representing spatio-temporal assertions is to restrict the class of expressible constraints to those of a *simple linear program* (SLP), with a view to exploiting known algorithms which are enormously simpler than those of full linear programming. Recall that an SLP is a collection of simple linear inequalities of the form:

$$q_i - q_j \geq a_{ij}$$

which, holding over pairs of objects, are known to be ideal for constraint propagation.

As evidence to support the claim of adequacy for simple linear inequalities, listed are all the example English statements taken from precisely the Malik/Binford paper just discussed:

1. The gas leak started immediately after takeoff.
2. John saw Mary a while ago.
3. My fever lasted 3 days.
4. A few days back, I was in Las Vegas.
5. Hiroshima was bombed on August 6, 1945.
6. I will finish my PhD in two to three years.
7. Jack had an accident a month after getting to Boston.
8. The symptoms start appearing within 10-20 minutes of the snakebite.

The authors note that "except for (2) and and possibly (4) all the specifications are linear relations between the endpoints of events ..." Each of these examples (except 2 and 4) is expressible by *simple* linear inequalities, since they involve no more than two variables each.

The single-source longest-path algorithm [12, 13, 14] is essentially a constraint-propagation technique, and determines in roughly *EV* time the consistency of an SLP.[10] A by-product of the algorithm is an assignment of values to the variables. This approach is much less complicated than the Simplex algorithm, easy to understand, dispenses with the specialized data structures of the latter, and is computable in polynomial time. Nogood sets are found by a depth-first search for a cycle starting from an edge (constraint) known to be unfulfilled when the algorithm terminates, and is therefore of order E for each nogood set returned. Furthermore, an SLP can support search by recursively marking vertices as

---

[9]MACSYMA is a full-blown mathematical manipulator, which can integrate, differentiate, handle algebraic expressions etc.

[10]Actually books on algorithms refer to shortest-path; reversing the signs gives the longest-path algorithm. There is not a unique single-source shortest-path algorithm. The cited Liao/Wong paper on VLSI layout, in which constraints of the lower-bound type preponderate, exploits the structure of the problem to improve the complexity to Emin(L,V), where L is the number of inequalities of the upper-bound type. At any rate, these algorithms favor sparse graphs, because for a complete graph the complexity "degrades" to a less impressive V^3.

8.2. An Abstract Formulation

Suppose that each dimension of interest in a task is sliced up into "slots", and that the task consists of allocating dimensional slots to certain objects, under some constraints specified explicitly by the task. These slots are identifiable with positive integers. A constraint is exactly of the form:

$$q_{id} - q_{jd} \geq a_{ij}$$

where the subscripted *d* indicates some dimension, the first subscript distinguishes between objects, and a_{ij} is any integer. Note that no constraint may mention more than two objects nor more than one dimension, in order to ease the implicit handling of the disjointness constraint. Sets of such constraints form a *simple linear program* (SLP). In words, they permit bounding the "distance" between two objects from above and below.

Worldly dimensional reasoning requires that a slot in time/space be filled by no more than one object. Call this requirement the *disjointness constraint*. Formally stated over a set of dimensions DIMS, this constraint becomes:

$$\forall(obj_i, obj_j \in OBJS)\ \exists(d \in DIMS)\ obj_{id} > obj_{jd} \vee obj_{id} < obj_{jd}$$

Tasks exhibiting constraints of these types are common: the space planning literature of a decade ago [22]; current efforts at VLSI layout and automatic schematics drafting.[12]

One could express the disjointness constraint *explicitly*, for all pairs of objects in the domain, and perform an exponential search through the user-specified constraints thrown in with these disjunctions. However, there is a better approach.

A multi-dimensional SLP together with the disjointness constraint is naturally captured by the *all-pairs* longest-path algorithm, which is a special case of the transitive closure as formulated by Aho, Hopcroft, and Ullman [12]. In this case, consistency holds if the following two types of constraint hold: the *transitive* constraint, and the *disjointness* constraint. A transitive contradiction occurs when, by tracing a chain of inequalities, one obtains the absurdity $q_i > q_i$. A disjointness contradiction happens when, colloquially, two things are in the same place at the same time, or formally that two quantities cannot assume the same value over each of the relevant dimensions.

The idea here is that one computes the longest-path between all pairs of objects (vertices in a graph), after which transitive contradictions cause positive-sum cycles to appear, where the weights of edges are the a_{ij} from (). Furthermore, a zero-sum path, existing in both directions between a pair of vertices, reveals what I shall call an *alignment*. An alignment indicates that two vertices are constrained to be equal in value, because each is required to be ≥ the other. If a pair of vertices *align* over each dimension from some prohibited set of dimensions, then a *clash*, or disjointness contradiction, occurs.

If the longest paths are stored as a matrix, then there is a positive integer along the matrix diagonal if and only if there is a transitive contradiction in the SLP. This calculation is linear in the number of vertices. An *alignment* is detected in time quadratic in the number of vertices, and further checking for alignments over all dimensions yields, as a whole, a complexity of $DV^2\log(V)$,[13] where D is the number of dimensions.

---

[12]Further back, the FINDSPACE problem [23] dealt with disjointness constraints in continuous space, for the problem of proposing a place for a new polyhedron among a group of already-situated polyhedra.

[13]There are at most V^2 alignments, and the intersection over the dimensions accounts for the logarithmic term.

possibly will be changed upon the addition of new constraints, and working only on these vertices. The author has implemented such a program and verified the qualities reported here. Appendix II discusses the program in more detail.

For example, within the Hardware Troubleshooting group at the MIT AI Lab, a program like this is useful for the task of creating a test program for digital circuits [15]. A feasible test program must satisfy the constraint between the clock initialization time t_{init} and any event t_e enabled by a clock pulse:

$$t_e - t_{init} \geq 1$$

A consistent assignment to the variables fulfilling this and other numeric constraints constitutes a candidate test program for some complex device.

This is yet another instance of the common trade-off between expressibility and computational tractability; circumscribing the possible constraints in this manner permits gains elsewhere.

8. Reasoning Under *Disjointness* Constraints

In the rest of this paper I advocate a new way to handle reasoning over space and time which is suitable under certain conditions to be discussed further. The proposal involves strictly limiting the expressivity of constraints in order to assure known favorable computational properties. This is a common and laudable theme underlying recent AI research [16, 17], since most AI systems involve a trade-off which often remains unclarified in the papers which report on them. Moreover, the proposal incorporates naturally a ubiquitous constraint in real problems: no two objects may occupy the same space at the same time.

8.1. Tasks Which Exhibit Disjointness

First I shall motivate what follows by describing domains which lend themselves to such an approach. There is currently an active effort aimed at automating the layout of circuit schematics [18, 19], flow diagrams [20], and integrated circuits [14]. Each of these tasks involves placing objects in a continuous plane of indefinite extent, and routing connections between the objects. The straightforward approach, common to each, "discretizes" the plane into a grid, so that placing an object involves assigning it to one of a finite set of slots.[11] One advantage, among many, of grids is that the connections have safe channels in which to navigate toward their attached objects, assured of not hitting any objects in the channels.

Generally the layout task is halved into *placement* and *routing*, in that order. Placement becomes the thematic subtask of assigning N objects to M slots. Random assignments will not do; typically one posts constraints that the layout must meet, and creates goals that have varying degrees of satisfaction. An implacable physical constraint which is not always made explicit in the constraint-satisfaction process is that no two objects may occupy the same position. Unfortunately the posting of constraints, during the search for a feasible layout, may result in the obligatory *co-incidence* of two objects. It is desirable to detect this as a by-product of the constraint-satisfaction process.

---

[11] An exception to this approach is the Design Problem Solver of a decade ago [21]. This program placed objects in a finite planar space by sequentially adjoining the objects to corners of the "room", or to already placed objects.

Since the all-pairs longest-path algorithm is cubic in complexity, I have shown that the consistency of a multi-dimensional simple linear program, together with the disjointness constraint, is soluble with complexity DV^3. Furthermore, such a beast is *useful* to many AI problem solving tasks.

Upon discovering a contradiction, it is easy to construct a nogood set of inequalities. The Warshall-Floyd version of the algorithm [24] is a special case of the transitive closure, and the latter works by methodically finding a "better" path from i to j that goes through some k. Whenever a better path is found, one records the "best" first step involved in going from i to j, which is the same as the current best first step from i to k. A nogood is built by tracing through these records which are a by-product of the transitive closure, *without doing any further search.*

It is possible to incorporate clustering into this approach, at the cost of doubling the number of prohibited sets of dimensions. If **P** is a set of dimensions over which it is forbidden to clash, then dividing a dimension $d \in \mathbf{P}$ into two clusters d_1 and d_2 means that no clash can occur on either of the sets

$\mathbf{P} - \{d\} + \{d_1\}$
$\mathbf{P} - \{d\} + \{d_2\}$.

One can fiddle with the expressions of complexity to determine whether this is worthwhile.

The program implemented in line with this work is more fully described in Appendix III. To further substantiate the claim of suitability, Appendix IV examines the incremental properties of this approach. Incremental properties are critical, because problem-solving typically involves *tentative* search, so the expedient addition and deletion of constraints is a virtue.

9. Conclusion

The key ideas of this paper are:

- Assertions that refer to space and time can in certain cases be represented similarly, with advantage.
- Allen's networks of temporal constraints are adequate for making inferences, but there is little hope of guaranteeing their consistency, short of exponential search through the disjunctions.
- Felicitously, this exponential search can be conducted in the same representation, through *singleton* networks, whose closure is provably consistent.
- All loops in a closed non-null Allen network include the element "=", and this theorem is useful as a heuristic to prune branches of the exponential search.
- Full linear programming and the Simplex algorithm, unshelved every few years by AI researchers, is not very satisfactory as a component of problem-solvers.
- A simple linear program (SLP) is a representation possessing simple known algorithms that determine consistency. The tradeoff is a reduced expressivity, compared with full linear programming. The properties of these known algorithms make them amenable to problem-solving: they are highly incremental, they support the reporting of nogood sets, and their data structures are object-oriented, and therefore integrate easily with other data structures within the global task.
- Layout tasks exhibit the tacit constraint of *disjointness*. By limiting explicit posted constraints to those of an SLP, all of these constraints are captured by the all-pairs longest-path algorithms. The incremental properties of these algorithms are satisfactory.

Acknowledgment

I wish to thank Professor Peter Szolovits for originally encouraging me to prepare this work as a memo. The results of discussions with Prof. Randall Davis, Walter Hamscher, Prof. Charles Leiserson, Mark Shirley, and Marc Vilain appear here in one form or another. Finally, I would also like to thank, on behalf of future readers and myself, to Mark Shirley for explaining a graduate seminar and thereafter giving its weak points.

I. Allen's Transitivity Table

This table is reproduced from Allen's ACM paper[4].

B r2 C / A r1 B	<	>	d	di	o	oi	m	mi	s	si	f	fi
"before" <	<	no info	< o m d s	<	<	< o m d s	<	< o m d s	<	<	< o m d s	<
"after" >	no info	>	> oi mi d f	>	> oi mi d f	>	> oi mi d f	>	> oi mi d f	>	>	>
"during" d	<	>	d	no info	< o m d s	> oi mi d f	<	>	d	> oi mi d f	d	< o m d s
"contains" di	< o m di fi	> oi di mi si	o oi dur con =	di	o di fi	oi di si	o di fi	oi di si	di fi o	di	di si oi	di
"overlaps" o	<	> oi di mi si	o d s	< o m di fi	< o m	o oi dur con =	<	oi di si	o	di fi o	d s o	< o m
"over-lapped-by" oi	< o m di fi	>	oi d f	> oi mi di si	o oi dur con =	> oi mi	o di fi	>	oi d f	oi > mi	oi	oi di si
"meets" m	<	> oi mi di si	o d s	<	<	o d s	<	f fi =	m	m	d s o	<
"met-by" mi	< o m di fi	>	oi d f	>	oi d f	>	s si =	>	d f oi	>	mi	mi
"starts" s	<	>	d	< o m di fi	< o m	oi d f	<	mi	s	s si =	d	< m o
"started by" si	< o m di fi	>	oi d f	di	o di fi	oi	o di fi	mi	s si =	si	oi	di
"finishes" f	<	>	d	> oi mi di si	o d s	> oi mi	m	>	d	> oi mi	f	f fi =
"finished-by" fi	<	> oi mi di si	o d s	di	o	oi di si	m	si oi di	o	di	f fi =	fi

FIGURE 4. The Transitivity Table for the Twelve Temporal Relations (omitting "=").

dur = {d s f} con = {di si fi}

II. Inequality Solver

II.1. Description

This describes the user-functions for a program that solves sets of simple linear inequalities of the form q_i-$q_j \geq a_{ij}$ where a_{ij} ranges over all integers.[14] "To solve" means to assign non-negative integers to the quantities, such that the inequalities are satisfied. An insoluble set of inequalities is detected by the program, and reported.

The inequalities are optionally grouped into bins (synonym: clusters) for reasons of efficiency, or by natural separation along dimensions. An example of a natural separation into bins is the case where each inequality holds over some one of the spatio-temporal dimensions, so that each bin represents a dimension.

This program is intended for use during *tentative* search, so for this reason it is highly incremental. Adding a few new constraints to a previously-solved set of constraints and then solving, takes on the average much less time than solving the entire new set of constraints. Internal data structures promote this incrementalism by remembering new constraints and enabling the recursive marking of quantities whose values *could* be affected by the new constraints.

If a set of constraints is solved from scratch, and proves consistent, then the time-order is E*min(L,V), where E is the number of current (not suspended) constraints, V is the number of quantities, and L is (approximately) the number of constraints whose a_{ij} is negative.

If the set of constraints proves inconsistent, then the time-order depends on the exhaustiveness of the information about nogoods. If only the first nogood set found is sufficient, then the order is

$$E+E*\min(L,V) = E*\min(L,V)$$

which is the same as before, but the constant of proportionality will be a little larger. For complete information about nogoods, the complexity is:

$$E*L + E*\min(L,V) = E*L$$

but the constant is considerably larger.

There is no smaller expression for the order in the incremental mode. However on the average it does considerably less work than solving from scratch.

Note: in this graph representation of the constraints, finding the complete relationship between two quantities involves determining the longest path between the two, possibly in either direction. This is not supported here. An alternative is to simply compare the assigned values of the quantities. However, from this comparison one can only conclude about possibility, not necessity (if one value is larger than the other, it may not be necessarily so). Necessity is determined only by the longest paths. If necessity is needed, then perhaps the alternative all-pairs longest-path approach is more appropriate.

---

[14] Actually $a_{ij} > -10^6$

II.2. External Functions

ACTION

1. (solve bin &optional (minimal-solution? nil) (find-nogood? t) (first-nogood? nil))

Solve attempts to find an assignment of values to the quantities such that no constraint is violated. The parameter <minimal-solution?> should be set true only when a "final" solution is desired, e.g. some further computation is done on the actual assignments to the quantities. A minimal solution is the minimal assignment in the positive integers which satisfies the inequal- ities. Solve returns nil if a consistent assignment is found, otherwise it returns nogood lists of "edges" (an internal representation of constraints) the number of which depends on the optional parameters. If <find-nogood?> is nil, on an inconsistency it returns simply a non-nil value.

INPUT

2. (Input-gte q1 q2 constant justifier bin)

Inputs the constraint q1 - q2 ≥ constant into bin, with given justifier.

3. (Input-lte q1 q2 constant justifier bin)

Inputs the constraint q1 - q2 ≤ constant into bin, with given justifier.

4. (same q1 q2 justifier bin)

Inputs the constraint q1=q2 into bin, with given justifier. This actually becomes two constraints: q1-q2≥0, and q1-q2≤0.

5. (nail q constant justifier bin)

Inputs the constraint q = constant into bin, with given justifier.

6. (Input-vertex q bin)

Informs of the existence of the input vertex. This function is useful when a vertex is not involved in any constraints with any other vertices, but its presence is needed anyway. For example, a vertex may need a number assigned to it, even if it is involved in no constraints.

DELETION

7. (empty &optional (bin nil))

Erases all the constraints from all bins, or from the optionally specified bin.

8. (delete-constraint q1 q2 weight justifier bin)

Deletes the constraint specified by the arguments.

INFORMATION

9. (any-constraints-violated? &optional (bin nil))

Returns nil if the current assignment of values to the quantities in <bin> violate no constraints. If no <bin> is given, all bins are tested. When there is an unfulfilled constraint in a bin, that bin is returned.

10. (constraint-justifier constraint)

Returns justifier of current constraint stored in <constraint>. The only legitimate way to obtain a <constraint> is through the nogood sets returned by the function *solve*.

11. (find-constraint q1 q2 bin)

Returns two values about the current (in force) constraint existing between the quantities <q1> and <q2>: the weight, and the justifier.

12. (quantity-value quantity bin)

Returns the value assigned to <quantity> in <bin>.

13. (collect-vertices bin)

Returns a list of the vertices in bin.

II.3. Program Source

If the reader is interested in a copy of the program described here, written in Common Lisp, he may contact the author at the institution on the cover, at the address 545 Technology Square; Cambridge, MA 02139; USA. Alternately, the Arpanet address is valdes@mit-htvax.

III. Reasoning with Disjointness Constraints

III.1. Description of the Program

The purpose of this program is to serve as a module which accepts sets of simple linear inequalities and determines whether those sets are consistent in two distinct ways.

Acceptable inequalities are exactly of the form $q_i - q_j \geq a_{ij}$, where the q's are quantities and a_{ij} is any integer > -10^6. Only two quantities are allowed per inequality. These inequalities are to be interpreted as constraints that must hold true for any assignment of values to the quantities.

Consistency holds if the following two types of constraint hold: the *transitive* constraint, and the *disjointness* constraint. A transitive contradiction occurs when, by tracing a chain of inequalities, one obtains the absurdity $q_i > q_i$. A disjointness contradiction happens when, colloquially, two things are in the same place at the same time, or formally that two quantities assume the same value over each of some specified set of bins (possibly interpreted as dimensions). I call the situation where two quantities are constrained to be equal an *alignment*, and the occurrence of alignments between the same quantities over some prohibited set of bins a *clash*. Disjointness constraints are not universal to all reasoning tasks, but are a physical reality and therefore arise often enough. This type of constraint is not smoothly incorporated into those algorithms which determine the consistency of a set of inequalities by constructing assignments to the quantities.

III.2. External Functions

ACTION

1. (clozure &optional (want-nogoods? t) (bins (get-all-bins)))

Determine whether the sets of constraints, over all bins, are consistent (contain no positive cycles nor clashes). Optionally returns some nogood sets.[15] Optionally executes the closure only for the specified bin(s), although clashes will still be detected in this case by intersecting the new alignments with those found previously for the other bins. Time order: B*V**3

INPUT

2. (add-ineq quantity1 quantity2 weight justifier bin)

Adds the inequality: quantity1 - quantity2 ≥ weight into <bin>, with given <justifier>. <weight> is any integer ≥ -10**6.

3. (identical quantity1 quantity2 justifier bin)

Constrain quantity1 and quantity2 to be equal, in <bin>, with given <justifier>. Equivalent to the two statements:

---

[15]A largest cycle-sum nogood set is returned for any quantity which is involved in a *transitive* contradiction. Also a nogood set is returned for any pair of quantities if: the pair forms a clash, *and* the pair are not both involved in a same transitive contradiction in any bin included in the set of bins over which the clash occurs. Usually a clash occurs over all the bins that the system knows about, but in general the user may specify multiple subsets of the full set of bins instead.

(add-ineq quantity1 quantity2 0 justifier bin) ;same justifier and bin
(add-ineq quantity2 quantity1 0 justifier bin) ;for both.

4. (initialize-bin bins &optional (size *initial-array-size*))

Informs about the existence of bins (or single bin). Whenever a constraint is input, a bin is created if necessary. However, a clash can be signalled spuriously on a set of input constraints if **clozure** is run *before* one of the bins is created by inputting a constraint. If this problem happens, just "declare" all the bins beforehand.

HYPOTHESIS

5. (hypothesize-alignment quantity1 quantity2 bin)

Examines whether an alignment in the indicated <bin> would cause <quantity1> and <quantity2> to clash. If affirmative, returns the set of bins over which the clash occurs. Does not modify the set of inequalities. Time order: B*log(V).

6. (hypothesize-ineq q1 q2 weight bin &optional (want-a-nogood? t))

Assuming that the current sets of inequalities are consistent (no transitive contradictions nor clashes), this function serves to hypothesize an inequality and detect whether its addition into <bin> would result in an inconsistency of either sort. It does *not* actually *add* the inequality to the set.

The function optionally returns a nogood (a list of justifiers), **MINUS** the justifier corresponding to the inequality which is hypothesized, upon finding an inconsistency. If a transit is found, then it doesn't bother checking for clashes also. If it checks for a clash and finds one, then a second value returned is a list of: the two quantities which *clash* plus the set of bins over which they clash (a list of length three). It returns nil if the new inequality is consistent.

The time order in the worst case is BV^3, although in practice this check is fast.

7. (introduces-clash? q1 q2 weight bin)

If the new inequality q1-q2≥weight added to bin causes a clash, then a list (i j bin-set) is returned where i,j (possibly different from q1,q2) are the pair that clash over the set of bins *bin-set*. Otherwise, it returns nil.

The time order in the worst case is $B*\log(V)V^2$, although in practice this check is fast.

DELETION

8. (delete-ineq quantity1 quantity2 weight justifier bin)

Deletes an inequality. This involves restoring a previously suspended inequality, if <weight> with <justifier> is the constraint currently in force (i.e. having the largest weight) between quantity1 and quantity2. Otherwise it just deletes the weight and justifier from the list of suspended weights.

9. (obliterate-bin &optional (bins (get-all-bins)))

Erases all cognizance of bins, which can be either a list or single symbol.

INFORMATION

10. (longest-path quantity1 quantity2 bin)

Returns the longest path (an integer) from quantity1 to quantity2 in bin.

11. (get-ineqs (&optional (bins (get-all-bins))))

Returns list of ineqs in optionally specified bin.

12. (aligned? quantity1 quantity2 bin)

Are the two input quantities aligned in the given bin?

13. (collect-alignments bin)

Constructs and returns a list of the current alignments in <bin>.

14. (find-ineq quantity1 quantity2 bin)

Returns two values: the weight and the justifier (possibly both nil) of the inequality currently in force between quantity1 and quantity2.

15. (bin-exists? bin)

Has the bin been created? Bins are created by add-ineq'ing an inequality with that bin as the destination.

16. (get-path quantity1 quantity2 bin)

Returns list of ineqs responsible for the longest path from quantity1 to quantity2. Time order: V.

17. (get-all-bins)

Returns list of current bins.

18. (quantity-in-bin? quantity bin)

Does the mentioned quantity exist in bin?

19. (neg-infinity? longest-path)

Asks whether <longest-path> is in fact minus infinity (it is represented internally as a large negative integer).

20. (ineq-source ineq) (ineq-sink ineq) (ineq-weight ineq) (ineq-bin ineq) (ineq-justifier ineq)

Self-evident.

III.3. Program Source

If the reader is interested in a copy of the program described here, written in Common Lisp, he may contact the author at the institution on the cover, at the address 545 Technology Square; Cambridge, MA 02139; USA.

IV. Incremental Properties

The claim here is that the all-pairs longest-path approach to *tentative* search in a domain with disjointness constraints is advantageous. If this claim is true, then the all-pairs algorithm must offer information about the effect of new constraints *cheaply*. Therefore, we examine how to extract such information, and at what cost, in the circumstance of the addition of a single inequality to a group (denoted by *bin*) of known contradiction-free inequalities.

In what follows let *lp(i,j,bin)* denote the current longest path from quantity *i* to quantity *j* in the referenced bin, and allow *ij* as shorthand for *lp(i,j,bin)* when the bin of reference is understood.

We add the single inequality

$$B - A \geq W_{A,B}$$

which creates a new longest path from A to B that is represented graphically as a link from A to B. One can determine whether that new constraint causes a positive cycle (transitive contradiction) in *constant* time.

Lemma 8: A transitive contradiction is introduced to a consistent set of constraints if and only if

$$W_{A,B} + BA > 0 \tag{2}$$

Proof by contradiction. Assume that (2) is false, and that a newly positive cycle

$$iA + W_{A,B} + Bi > 0 \tag{3}$$

exists which includes $W_{A,B}$. Since

$$Bi + iA \leq BA \tag{4}$$

because of the *definition* of BA (which is unchanged by the new inequality), we can combine the three previous inequalities and obtain a contradiction. This proves necessity. Sufficiency follows from the definition of a transitive contradiction.

Q.E.D.

Now assuming that no positive cycle was introduced by the new inequality, one can check whether that new inequality introduces an *alignment*. We focus on conditions necessary for the existence of a new alignment between any quantities *i* and *j*. This new alignment must include the new step AB, so by definition

$$\begin{aligned} &iA + W_{A,B} + Bj = 0 && \text{;definition of alignment(i,j)} \\ &ji = 0 && \text{;definition of alignment(i,j)} \\ &BA \geq Bj + ji + iA && \text{;definition of longest path} \end{aligned} \tag{5}$$

Since no positive cycle is present,

$$W_{A,B} + BA \leq 0 \tag{6}$$

From (5) and (6) it follows that

$BA = W_{A,B}$

so the round trip between A and B is zero, if an alignment exists.

To further determine whether the alignment between *i* and *j* in fact causes a clash involves the intersection of the new alignment with alignments in the other bins of the prohibited sets.

A formalization of the previous statements is due.

Theorem 9: The addition into *bin* of a new inequality

$B - A \geq W_{A,B}$

introduces a transitive contradiction to previously contradiction-free bins of inequalities if and only if

$W_{A,B} + lp(B,A,bin) > 0;$

it introduces a disjointness contradiction if and only if

$W_{A,B} + lp(B,A,bin) = 0$, *and*

$\exists(i,j)\ \{lp(j,i,bin) = 0 \ \wedge\ lp(i,A,bin) + W_{A,B} + lp(B,j,bin) = 0 \ \wedge$
would-clash-if-align$(i,j,bin)\}$

where

$alignment(k,l,bin) \Leftrightarrow lp(k,l,bin) = lp(l,k,bin) = 0$

would-clash-if-align$(k,l,bin) \Leftrightarrow$
$(alignment(k,l,bin) \Rightarrow$
$\exists(set \,||\, bin \in set)\forall(b \in set)$ if $b \neq bin$ then $(lp(k,l,b) = lp(l,k,b) = 0)$

The statement of the theorem suggests a procedure to compute whether a new constraint introduces a contradiction. The time order of this procedure is BV^2 times the time to look up whether a given alignment exists in a bin. Since alignments in this program are stored in a binary tree, the time becomes $BV^2\log(V)$, as a worst case. If the new inequality between A and B does not result in a zero round trip between A and B, then the fact of *no clash* is known in constant time. Otherwise the order above is correct.

This order is somewhat better than that of re-computing all the longest paths, which is BV^3, but more importantly the time constants of the procedure that the theorem suggests are small. This adequately substantiates the claim that this approach, borrowed from known graph-theoretic algorithms, promotes search by cheapening the incremental cost of addition of new constraints. Also, as discussed previously, the return of nogood sets is accomplished *without* an increase in the time order of execution.

In general, a deletion involves replacing the deleted weight by a smaller weight, because the larger may have supplanted the smaller. The smaller weight must be reactivated when the larger is deleted. This act of deletion cannot introduce an inconsistency in a consistent set of inequalities. This is because a stronger constraint is replaced by a weaker constraint, so any set of inequalities satisfying the first also

satisfies the second. No alignment can be introduced by a deletion, since that would mean that a positive cycle had existed before the deletion.

so the bound $lp(_{A,B}$ between A and B is zero; if an alignment exists

To further determine whether the alignment between $_{A,B}$ and $_{A,B}$ causes a cliash involves the intersection of the new alignment with alignments in the other bins of the partitioned sets

A formalization of the previous statements is due:

Theorem 3: The addition into bin of a new inequality

$$B - A \geq W_{A,B}$$

introduces a transitive contradiction to previously contradiction-free bins of inequalities if and only if

$$W_{A,B} + lp(B,A,bin) < 0;$$

it introduces a disjointness contradiction if and only if

$$W_{A,B} + lp(B,A,bin) = 0, \text{ and}$$

$$\exists(i,j)\ (lp(j,i,bin) = 0 \wedge lp(i,A,bin) + W_{A,B} + lp(B,j,bin) = 0 \wedge \underline{\text{would-clash-if-align}}(i,j,bin))$$

where

$$\text{alignment}(k,l,bin) \Leftrightarrow lp(k,l,bin) = lp(l,k,bin) = 0$$

$$\underline{\text{would-clash-if-align}}(k,l,bin) \Leftrightarrow$$
$$(\text{alignment}(k,l,bin) \Rightarrow$$
$$\exists(\text{set}_i \text{ bins set})\forall(\text{be set}_i)\ \text{if be} \neq bin \text{ then } (lp(k,l,b) = lp(l,k,b) = 0))$$

The statement of the theorem suggests a procedure to compute whether a new constraint introduces a contradiction. The time order of this procedure is BV^2 times the time to look up whether a given alignment exists in a bin. Since alignments in this program are stored in a binary tree, the time becomes $BV^2 \log(V)$, as a worst case. If the new inequality between A and B does not result in a zero bound trip between A and B, then the test of no clash is known in constant time. Otherwise the order above is correct.

This order is somewhat better than that of re-computing all the bridges paths, which is BV^3 but more importantly the time constants of the procedure that the theorem suggests are small. This substantiates the claim that this approach, borrowed from known graph-theoretic algorithms, promises search by cheapening the incremental cost of addition of new constraints. Also, as discussed previously, the return of negated sets is accomplished without an increase in the time order of execution

In general, a deletion involves replacing the deleted weight by a smaller weight, because the larger may have supplanted the smaller. The smaller weight must be recovered when the larger is deleted. This act of deletion cannot introduce an inconsistency in a consistent set of inequalities. This is because a stronger constraint is replaced by a weaker constraint, so any set of inequalities satisfying the first also

References

1. D. Marr. "Artificial Intelligence - A Personal View". *Artificial Intelligence 9* (1977), 37-48.

2. B.L. Whorf. An American Indian Model of the Universe. In *The Philosophy of Time*, R.M. Gale, Ed., Humanities Press, 1968.

3. R.J. Brachman, R.E. Fikes, and H.J. Levesque. "Krypton: A Functional Approach to Knowledge Representation". *IEEE Computer 16*, 10 (Oct 1983), 67-73.

4. J.F. Allen. "Maintaining Knowledge about Temporal Intervals". *Communications of the ACM 26*, 11 (1983), 832-843.

5. J.F. Allen. An Interval-Based Representation of Temporal Knowledge. Proceedings of IJCAI-7, 1981, pp. 221-226.

6. J.F. Allen and J.A. Koomen. Planning Using a Temporal World Model. Proceedings of IJCAI-8, 1983, pp. 741-747.

7. U. Montanari. "Networks of Constraints: Fundamental Properties and Applications to Picture Processing". *Information Sciences 7* (1974), 95-132.

8. J.F. Allen and P.J. Hayes. A Common-Sense Theory of Time. Proceedings of IJCAI-9, 1985, pp. 528-531.

9. M.B. Vilain. . Personal Communication.

10. J. Malik and T. Binford. Reasoning in Time and Space. Proceedings of IJCAI-8, 1983, pp. 343-345.

11. C.L. Liu. *Introduction to Combinatorial Mathematics*. McGraw Hill, 1968.

12. A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

13. C.E. Leiserson and J.B. Saxe. A Mixed-Integer Linear Programming Problem Which is Efficiently Solvable. Proceedings of 21st Annual Allerton Conf. on Communications, Control, and Computing, 1983.

14. Y.Z. Liao and C.K. Wong. "An Algorithm to Compact a VLSI Symbolic Layout with Mixed Constraints". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems CAD-2*, 2 (April 1983), 62-69.

15. M.H. Shirley. An Automatic Programming Approach to Testing. IEEE Workshop on Test Generation Environments, 1985.

16. R.J. Brachman and H.J. Levesque. The Tractability of Subsumption in Frame-Based Description Languages. Proceedings of IJCAI-84, 1984, pp. 34-37.

17. P.F. Patel-Schneider. Small can be Beautiful in Knowledge Representation. Technical Report 37, Fairchild Lab for A.I. Research, 1984.

18. I.O. Tou. Automatic Formatting of Logic Schematics. Master Th., Massachusetts Institute of Technology,1984.

19. A. Kumar, A. Arya, V.V. Swaminathan, and A. Misra. "Automatic Generation of Digital System Schematic Diagrams". *IEEE Design & Test* (February 1986), 58-65.

20. L.B. Protsko, P.G. Sorenson, and J.P. Tremblay. A System for the Automatic Generation of Data Flow Diagrams. Research Report 84-8, Department of Computational Science, University of Saskatchewan, 1984.

21. C. Pfefferkorn. The Design Problem Solver. In *Spatial Synthesis in Computer-Aided Building Design*, C. Eastman, Ed., Halsted Press, 1975.

22. M. Henrion. Automatic Space-Planning: A Post-Mortem? In *Artificial Intelligence and Pattern Recognition in Computer Aided Design*, J.C. Latombe, Ed., North Holland, 1978.

23. G.J. Sussman. The FINDSPACE Problem. Memo 286, MIT Artificial Intelligence Laboratory, 1973. Out of Print..

24. N. Deo. *Graph Theory with Applications to Engineering and Computer Science.* Prentice Hall, 1974.

25. K. Kahn and G.A. Gorry. "Mechanizing Temporal Knowledge". *Artificial Intelligence 9* (1977), 87-108.

26. I. Munro. "Efficient Determination of the Transitive Closure of a Directed Graph". *Information Processing Letters 1* (1971), 56-58.

27. A. Mackworth. "Consistency in Networks of Relations". *Artificial Intelligence 8* (1977), 99-118.

28. E.Yu. Kandrashina. Representation of Temporal Knowledge. Proceedings of IJCAI-8, 1983, pp. 346-348.

29. M.B. Vilain. A System for Reasoning About Time. Proceedings of AAAI, 1982, pp. 197-201.

30. D.B. Johnson. "Efficient Algorithms for Shortest Paths in Sparse Networks". *Journal of the ACM 24*, 1 (Jan 1977), 1-13.

CS-TR Scanning Project
Document Control Form

Date : 11/9/95

Report # AIM-875

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
☐ Laboratory for Computer Science (LCS)

Document Type:

☐ Technical Report (TR) ☒ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 39 (40 - IMAGES)
Not to include DOD forms, printer instructions, etc... original pages only.

Originals are:
☐ Single-sided or
☒ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☒ Laser Print
☐ InkJet Printer ☐ Unknown ☐ Other:_____

Check each if included with document:

☒ DOD Form (2) ☐ Funding Agent Form ☐ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number):_____

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description :	Page Number:
IMAGE MAP: (1-34) UNNUMBERED TITLE PAGE,	1-33
(35-40) SCANCONTROL, DOD (2), TRGT'S (3)	

Scanning Agent Signoff:
Date Received: 11/9/95 Date Scanned: 11/28/95 Date Returned: 11/30/95

Scanning Agent Signature: Michael W. Cook

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: Memo 875	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER: AD-A176659	
4. TITLE (and Subtitle): Spatio-Temporal Reasoning and Linear Inequalities	5. TYPE OF REPORT & PERIOD COVERED: Memo
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Raúl E. Valdés-Pérez	8. CONTRACT OR GRANT NUMBER(s): N00014-85-K-0124
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Artificial Intelligence Laboratory, 545 Technology Square, Cambridge, MA 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Advanced Research Projects Agency, 1400 Wilson Blvd., Arlington, VA 22209	12. REPORT DATE: May 1986
	13. NUMBER OF PAGES: 33
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office): Office of Naval Research, Information Systems, Arlington, VA 22217	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

None.

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Spatial Reasoning
Temporal Reasoning
Constraint Networks
Problem-Solving
Layout

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Time and Space are sufficiently similar to warrant in certain cases a common representation in AI problem-solving systems. What is represented is often the constraints that hold between objects, and a concern is the overall consistency of a set of such constraints.

(over)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Block 20 -

This paper scrutinizes two current approaches to spatio-temporal reasoning. The suitableness of Allen's temporal algebra for constraint networks is influenced directly by the mathematical properties of the algebra. These properties are extracted by a formulation as a network of set-theoretic relations, such that some previous theorems due to Montanari apply. Some new theorems concerning consistency of these temporal constraint networks are also presented.

It is argued that the linear programming approach to reasoning in time and space is needlessly complex, and is otherwise unsuitable as a submodule for a task that performs search.

In the spirit of the thematic tradeoff between expressivity and tractability, simple linear inequalities are adequately expressive and provide known algorithms which are amenable to a search regimen. Moreover, another known algorithm captures the real physical constraint of disjointness, and is therefore relevant to layout and planning tasks.

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**